**Contemporary Engineering Sciences, Vol. 7, 2014, no. 23, 1287 - 1293**
**HIKARI Ltd, www.m-hikari.com**
**http://dx.doi.org/10.12988/ces.2014.49159**

# Achieving Fast Computer-Generated Hologram Calculations via Parallelization

**Hyun Jun Choi**

Department of Electronic Engineering
Mokpo National Maritime University
Jeonnam, 530-729, Korea

**Dong Kwan Kim**

Department of Computer Engineering
Mokpo National Maritime University
Jeonnam, 530-729, Korea

## Abstract

Computer-Generated Holography (CGH) plays an important role in the development of three-dimensional display. However, the enormous computational time for CGH generation hinders the practicality of the CGH—depending on a specific application, it takes several hours, even more to complete CGH computations. To enhance the hologram computation speed, we present a parallel computing approach to accelerating long-running computations of the CGH application to provide high resolution digital holograms. A typical sequential version of a CGH program is parallelized by extracting independent tasks. Message Passing Interface (MPI) is used to communicate with parallel tasks and an entire computation will be assigned among different computational nodes. For all the experiments, we will use a supercomputer which is a 60-node heterogeneous CPU/GPU cluster. To validate our approach, we have applied our parallel version of the CGH to generate a digital hologram from a 3D object. The experimental results demonstrate that a variety of parallelization strategies could be a novel way to increase computation speed improvement for computationally-intensive CGH applications.

**Keywords**: Digital Hologram, Parallelization, High-End Computing, Message Passing Interface, Parallel Program

## 1 Introduction

Digital hologram can be generated by two methods. First, an interference pattern that was recorded on holographic analog film can be recorded as digital data using a CCD camera. Second is the computer generated hologram technique. The generated digital hologram is displayed on the spatial light modulator (SLM) of the receiver and the 3D image is reproduced spatially by scanning with laser light. Optical equipment is high priced, it takes considerable effort to handle them in order to obtain or display images and even CGH requires complex operations to get the holograms. Therefore, the need to protect such high end contents like holograms is rising.

Recent trend of the 3D image/video is heading to a perfect realistic 3D without restriction in viewing angle, wearing glasses, etc. It can be realized by hologram [1] which has been gathering more attention recently by that reason. However, in order to achieve large image size with wide viewing angle and full parallax effect, holograms with large pixel count are required [2]. Generation of such holograms is computationally expensive and remains one of the main challenges to make holographic 3D display technology.

As parallel programming is growing rapidly under multicore processors, we can expect such parallel computing can contribute to the speed improvement for the CGH computation. We propose a parallel computing approach to accelerating long-running computations of the CGH application to provide high resolution digital holograms.

This paper is organized as follows. Section 2 introduces the CGH technique, and Section 3 explains the proposed algorithm. The implementation results of the proposed algorithm are described in Section 4, and based on this, Section 5 presents the conclusion.

## 2 Computer Generated Hologram

CGH has the ability to correctly record and reconstruct a light wave for a 3D object. Assuming that a 3D object is composed of N point light sources, the formula for computing a CGH is expressed as [3]:

$$I_\alpha = \sum_j^N A_j \cos(k\sqrt{(px_\alpha - px_j)^2 + (py_\alpha - py_j)^2 + z_j{}^2}) \qquad (1)$$

where, $\alpha$ or $j$ indicates a particular point on the hologram or 3D object respectively, $k$ is the wave number of the reference wave defined as $2\pi/\lambda$, p represents the pixel pitch of the hologram, and ($x_\alpha$ , $y_\alpha$) and ($x_j$, $y_j$, $z_j$) represent the coordinates of the hologram and 3D object respectively.

Figure 1 shows a sample coordinate array system for the 3D object and the digital hologram to apply the CGH method, where the 3D object consists of 2×2, and the digital hologram is captured as 4×4 in size. To generate a digital hologram with this set-up, the calculation of Eq. (1) must be carried out 64 (=2×2×4×4) times.

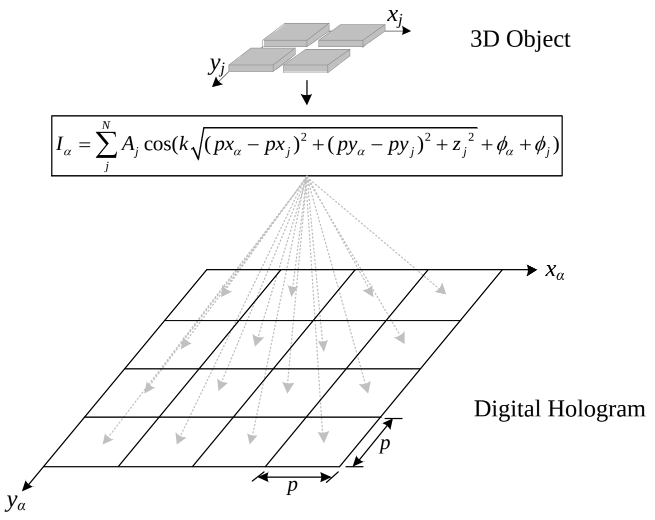

$x_j$
$y_j$
3D Object
$I_\alpha = \sum_j^N A_j \cos(k\sqrt{(px_\alpha - px_j)^2 + (py_\alpha - py_j)^2 + z_j{}^2} + \phi_\alpha + \phi_j)$
$x_\alpha$
Digital Hologram
$p$
$p$
$y_\alpha$

Fig. 1. Computation method of digital hologram.

## 3 Approach Overview

In this paper, loop parallelization [4, 5] is used to parallelize a sequential version of the CGH program. A loop can be classified into two categories according the execution scheme viz. sequential loops and parallel loops. The sequential loop means that the “i”-th iteration can be started after the “i−1”-st loop iteration is completed. In contrast to the sequential loop, the parallel loop can be executed in arbitrary order if there are no dependencies between the iterations of a loop. Therefore, in the parallel loop, we cannot guarantee that the "i-1"-st loop iteration will complete before the "i"-th one. There can be a variety of the parallel loops depending on the coding style. Shown in Figure 1, the CGH algorithm performs 2×2 array computations by iteratively traversing a large data structure. In particular, the CGH program contains nested `for` loops that take up the most execution time during the computation. We have transformed these loops parallel so that they can be executed independently on different processors.

As mentioned above, CGH provides a relatively convenient way to record and reconstruct a light wave for a 3D object. We assume that a 3D object is composed of N point light sources. The entire digital hologram can be produced by merging each digital hologram which is constructed from the corresponding light source. The CGH calculation for each light source can be performed in parallel since it is an independent task. These processes are required a substantial amount of calculation. For example, when we create a digital hologram of 1,024×1,024 pixels from a 3D object of 200×200 pixels, it needs a total of 41,943,040,000 calculation times. Figure 2 describes an example which generates a digital hologram from a 3D object according to the CGH formula. This example will be used for our case study.

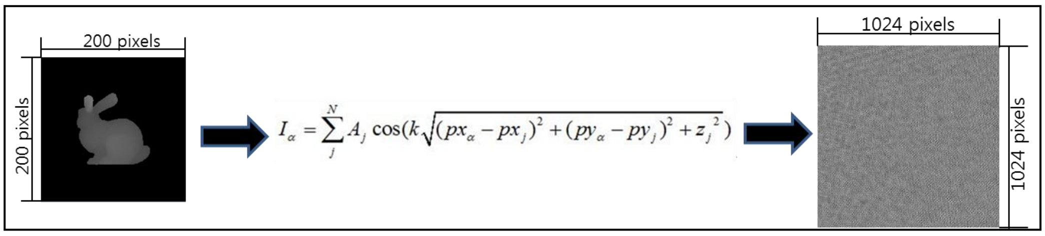
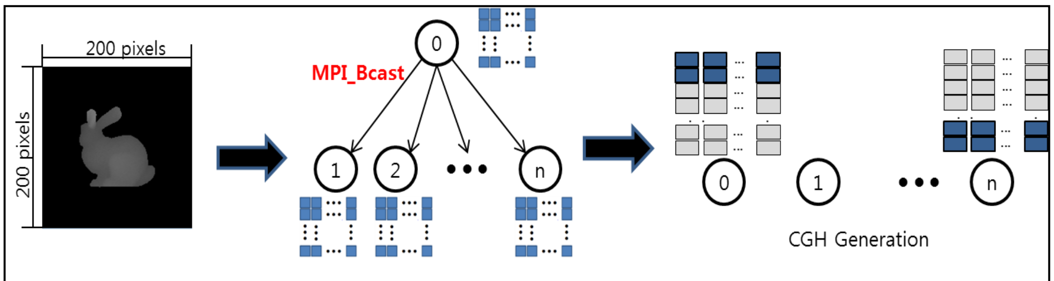

200 pixels
200 pixels
$I_\alpha = \sum_j^N A_j \cos(k\sqrt{(px_\alpha - px_j)^2 + (py_\alpha - py_j)^2 + z_j^2})$
1024 pixels
1024 pixels

Fig. 2. Generating a digital hologram from a 3D object using the CGH formula.

## 4 Implementation

Our approach aims at maximizing the degree of parallelism to enhance the efficiency of the CGH computation. We have designed and implemented a Message Passing Interface (MPI) application to show the applicability of parallel computing in digital holographic systems. MPI is a standardized library specification for message-passing on both massively parallel machines and on workstation clusters [6, 7]. As our implement environment, we have used a computer cluster which consists of a 60-node heterogeneous CPU/GPU cluster. Each compute node of the cluster runs two Intel Xenon processors with 8 cores each (a total of 16 cores), Four GPUs, 128GB RAM, and Red Hat Enterprise Linux 6.3. The nodes are connected by InfiniBand (40Gbit+).

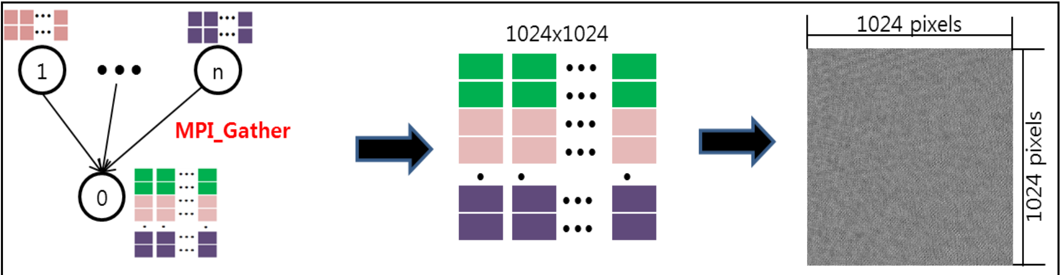
Figures 3 and 4 depict the overall flow of the proposed parallel computation—sharing data, partitioning tasks, and collecting the results of cluster nodes. Figure 2 shows how data sharing occurs on the computer cluster. The root node, process 0 in Figure 3, in the MPI program makes a two-dimensional image array to represent a three-dimensional object and then sends the image array to all participant processes(in Figure 3, process 1 to n) using the `MPI_Bcast` collective operation. `MPI_Bcast()` is one of the standard collective communication techniques and is used to send out the same data to a parallel program, or to all processes. Although the root process and receiver processes do different jobs, they all call the same `MPI_Bcast` function in Figure 3. When the root process calls `MPI_Bcast`, 40,000 integers will be broadcast from process 0 to every process in the group. When all of the participating processes call `MPI_Bcast`, the `Image` variable will be filled in with the image data from the root process. The `Image` variable is a two-dimensional array with 200 rows and 200 columns. Each process including the root process will perform the same CGH computation against different data set of the rabbit image.



```
MPI_Bcast(Image, width*height, MPI_INT, 0, MPI_COMM_WORLD)
```

Fig. 3. Sharing the two-dimensional array to represent the three-dimensional object among the cluster nodes using the `MPI_Bcast` collective operation.

Figure 4 shows how the `MPI_Gather` operation works after every process has completed its CGH computation. The root process has a valid receive buffer called `Finge_P` to store the computation results. Once all CGH computations have completed, `MPI_Gather` takes individual computation results from all processes in the group and then gathers them to the root process. The root process generates an entire digital hologram from the collected data.



```
MPI_Gather(&Fringe_Pattern[from], Fringe_width*Fringe_height/numnodes, MPI_INT,
          &Fringe_P, Fringe_width*Fringe_height/numnodes, MPI_INT, 0, MPI_COMM_WORLD)
```

Fig. 4. Generating the digital hologram by gathering each computation result from the participating node using the `MPI_Gather` collective operation.

As seen in Figure 5, we measured the execution time for six cases by varying the number of participating cluster nodes. As expected, the execution time tends to speed up as the number of cluster nodes increases. The sequential version of the

CGH program takes 1,435.43 seconds to calculate the same computation. Such results show we can achieve outstanding performance improvement when using parallelization.

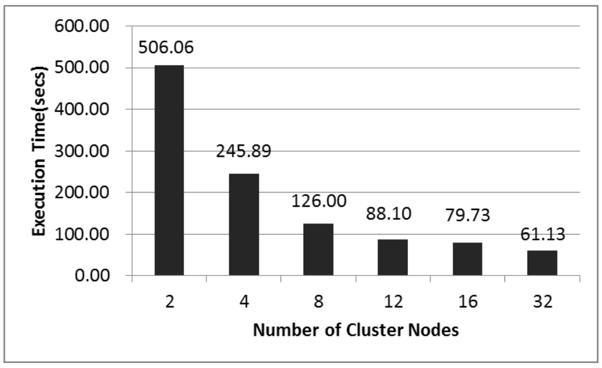


Fig. 5. Execution time of calculating computer-generated holograms in the compute cluster. x-axis: the number of nodes in the computer cluster, y-axis: the total execution time .

## 5 Conclusions

In this paper, we have considered the value of using parallel computing to speed up the computer-generated hologram computation. We have presented on our own work on parallelizing an existing sequential CGH program by using loop parallelism. The proposed approach finds independent iterations of a time-consuming loop and executes them in parallel by different processors. We also demonstrated the benefits of parallel computations for the digital holographic domain.

Along with parallel computing, dynamic software updating (DSU) [8, 9] can be another novel way to efficiently solve a computationally-intensive problem. DSU is a software engineering approach for making changes to software systems while they run. DSU can be effectively applied to parallel computer-generated holography. Therefore, it is worth exploring dynamic software updating as future work to achieve fast generation of digital video holograms.

**Acknowledgements.** This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2012R1A1A1006022, NRF-2010-0022080).

## References

[1] S. A. Benton and V. M. Bove Jr., Holographic Imaging, John Wiley & Sons, Inc., Hoboken, NJ (2008).

[2] X. W. Xu, S. Solanki, X. A. Liang, Y. C. Pan, and T. C. Chong, "Full high-definition digital 3D holographic display and its enabling technologies," Proc. SPIE 7730, 77301C (2010).

[3] H. J. Choi, Y. H. Seo, S. W. Jang, and D. W. Kim, "Analysis of Digital Hologram Rendering Using a Computational Method," Journal of information and communication convergence engineering, vol. 10, no. 2, pp. 205--209 (2012).

[4] T. Rauber and G. Rünger, Parallel Programming for Multicore and Cluster Systems, Springer, (2013).

[5] A. Beletska, W. Bielecki, A. Cohen, M. Palkowski, and K. Siedlecki, Coarse-grained loop parallelization: Iteration Space Slicing vs affine transformations, Parallel Computing, vol. 37, issue 8, pp. 479-497. (2011)

[6] The Message Passing Interface (MPI) standard, http://www.mcs.anl.gov/research/projects/mpi/.

[7] Message Passing Interface (MPI) Tutorial, https://computing.llnl.gov/tutorials/mpi/

[8] D. K. Kim, E. Tilevich, and C. J. Ribbens, Dynamic Software Updates for Parallel High Performance Applications. Concurrency and Computation: Practice and Experience, vol. 23, issue 4, pp. 415--434. (2010).

[9] C. M. Hayden, E. K. Smith, M. Denchev, M. Hicks, and J. S. Foster. Kitsune: Efficient, General-purpose Dynamic Software Updating for C, ACM Conference on Object-Oriented Programming Languages, Systems, and Applications, October (2012).

**Received: August 18, 2014**